ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ VIỄN THÔNG

BÁO CÁO MÔN HỌC THỰC HÀNH
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

NỘI DUNG:
GIẢI THÍCH CHI TIẾT GIẢI THUẬT VÀ CÁCH LẬP TRÌNH CÁC BÀI TOÁN
MÃ ĐI TUẦN, XẾP HẬU VÀ SUDOKU.

GIẢNG VIÊN HƯỚNG DẪN: ThS. CAO TRẦN BẢO THƯƠNG
SINH VIÊN THỰC HIỆN:

Trần Công Huy	**18200129**
Võ Văn Chính	**18200070**
Nguyễn Ngọc Anh Hào	**18200095**

Thành phố Hồ Chí Minh, ngày 25 tháng 01 năm 2022

Mục Lục

1. Bài toán Sudoku .

1.1. Giới thiệu game Sudoku.

- Sudoku có tên gọi tiếng Anh là Number Place, là một trò chơi câu đố sắp xếp chữ số dựa trên logic theo tổ hợp.
- **Mô tả:** Cho ma trận 9x9 trong đó có 9 ma trận con 3x3. Mục tiêu của trò chơi là điền các chữ số vào một ma trận 9×9 sao cho mỗi cột, mỗi hàng, và mỗi phần trong số chín lưới con 3×3 cấu tạo nên lưới chính đều chứa tất cả các chữ số từ 1 tới 9
- Luật chơi:
  + Điền các chữ số từ 1 đến 9 vào 9 ma trận 3x3 sao cho không có số nào bị trùng nhau
  +Trên cùng một hàng ngang trong ma trận 9x9 không được có 2 số bị trùng nhau
  +Trên cùng một hàng dọc trong ma trận 9x9 không được có 2 số bị trùng nhau

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Một bảng Sudoku điển hình

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Cách giải của bảng trên

1.2. Giải thuật game Sudoku.

*** Sử dụng thuật toán quay lui – Backtracking cho trò chơi Sudoku**
- Thuật toán quay lui - Backtracking là một giải thuật dựa trên đệ quy, là một giải thuật được sử dụng phổ biến trong khoa học máy tính.
- Quay lui là một kĩ thuật thiết kế giải thuật dựa trên đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn khả dĩ và đệ quy. Người đầu tiên đề ra thuật ngữ này (backtrack) là nhà toán học người Mỹ D. H. Lehmer vào những năm 1950.
- **Thuật toán:** Dùng để giải bài toán liệt kê các cấu hình tương tự như Sudoku. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng.

- Các bước của thuật toán có hình dạng X[1, … ,n]:

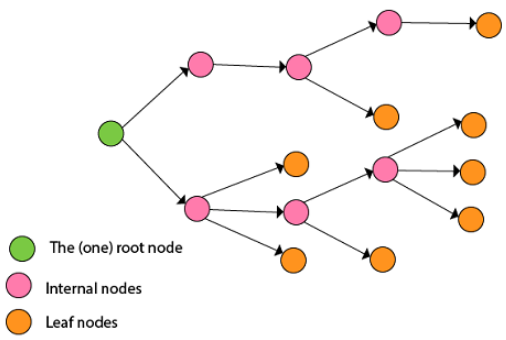
B1. Xét tất cả giá trị của X[1] so với điều kiện có thể nhận,

B2. Với mỗi giá trị X[1] ta sẽ tiếp tục tìm giá trị X[2] tiếp theo thỏa điều kiện. Ứng với với mỗi giá trị của X[2] ta tiếp tục X[3] cho tới khi:

…

Bn. Xét tất cả giá trị của X[n] thỏa điều kiện có thể nhận và thông báo kết quả

=>Bản chất của quay lui là một quá trình tìm kiếm theo chiều sâu(Depth-First Search).



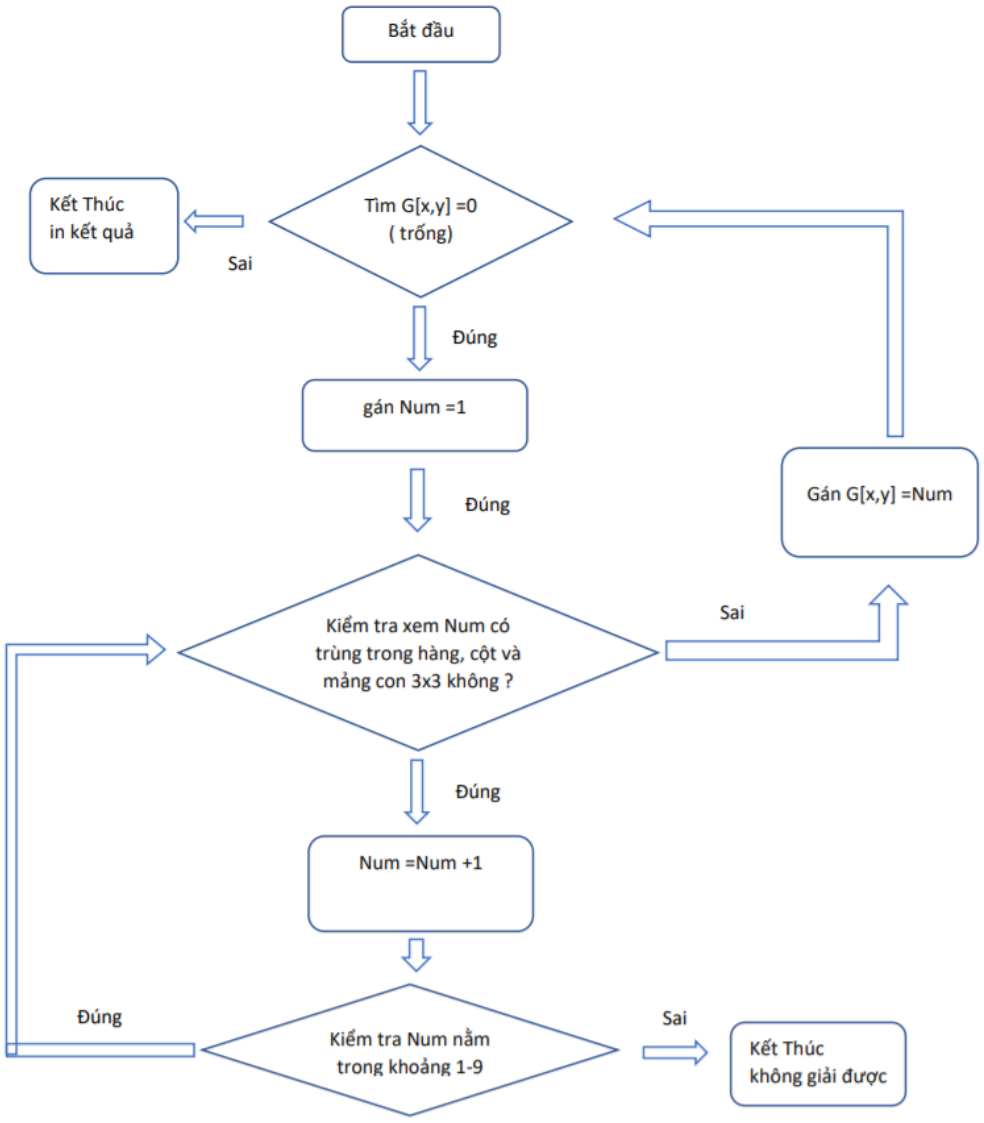
-Ưu điểm của thuật toán: Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy.

-Nhược điểm của thuật toán: Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ. Vì nó mắc phải các nhược điểm sau:

+ Thực hiện các công việc dư thừa: Mỗi lần chúng ta quay lui, chúng ta cần phải đánh giá lại lời giải trong khi đôi lúc điều đó không cần thiết.

+ Không sớm phát hiện được các khả năng bị bế tắc trong tương lai. Quay lui chuẩn, không có cơ chế nhìn về tương lai để nhận biết đc nhánh tìm kiếm sẽ đi vào bế tắc.

1.3. Sơ đồ giải thuật Game Sudoku



- Giải thích:

b1. Tìm ô trống, ô =0 , nếu không còn ô trống thì kết thúc chương trình

b2. Gán Num =1

b3. Kiểm tra Num có trùng Hàng , Cột và mảng con 3x3 không?

b4. Nếu không trùng thì quay lại bước 1, nếu trùng thì gán Num = Num +1

b5. Sau đó kiểm tra xem Num có nằm trong khoảng 1 đến 9

b6. Nếu không trong khoản 1 đến 9 thì kết thúc thông báo không giải được. nếu nằm trong khoảng đó thì quay lại b3

1.4. Cách lập trình Sudoku

- Kiểm tra xem số có nằm trong cột không

```
bool checkCol(int col, int num) {
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

- Kiểm tra xem số có nằm trong hàng không

```
bool checkRow(int row, int num) {
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

- kiểm tra số có trùng trong mảng con 3x3

```
bool checkBox(int boxStartRow, int boxStartCol, int num) {
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row + boxStartRow][col + boxStartCol] == num)
                return true;
    return false;
}
```

- kiểm tra số không có trùng trong hàng, côt và mảng con 3x3

```
bool checkAll(int row, int col, int num) {
return !checkRow(row, num) && !checkCol(col,num)&& !checkBox(row-row%3,
            col-col%3,num);
    }
```

- In ra kết quả sau khi giải xong, in ra dạng lưới ngăn cách các ô 3x3

```
    void sudokuGrid() {
        for (int row = 0; row < N; row++) {
            for (int col = 0; col < N; col++) {
                if (col == 3 || col == 6)
                    cout << " | ";
                cout << grid[row][col] << " ";
            }
            if (row == 2 || row == 5) {
                cout << endl;
                for (int i = 0; i < N; i++)
                    cout << "---";
            }
            cout << endl;
        }
    }
```

- Tìm vị trí để trống

```
    bool findEmptyPlace(int& row, int& col) {
        for (row = 0; row < N; row++)
            for (col = 0; col < N; col++)
                if (grid[row][col] == 0)
                    return true;
        return false;
    }
```

- Giải thuật Sudoku

```
    bool solveSudoku() {
        int row, col;
        if (!findEmptyPlace(row, col))
            return true; //Khi tất cả ô được điền
        for (int num = 1; num <= 9; num++) {//Số hợp lệ nằm tron khoảng từ 1-9
            if (checkAll(row, col, num)) {
//Kiểm tra xe số có nằm trong hàng, cột và mảng con 3x3

                grid[row][col] = num;

                if (solveSudoku())
//đệ quy để tiếp tục tìm các giá trị tiếp theo
                    return true;

                grid[row][col] = 0;
// Khi không có điều kiện nào thỏa mãn gán bằng 0
            }
        }
        return false;
    }
```

- kết quả:

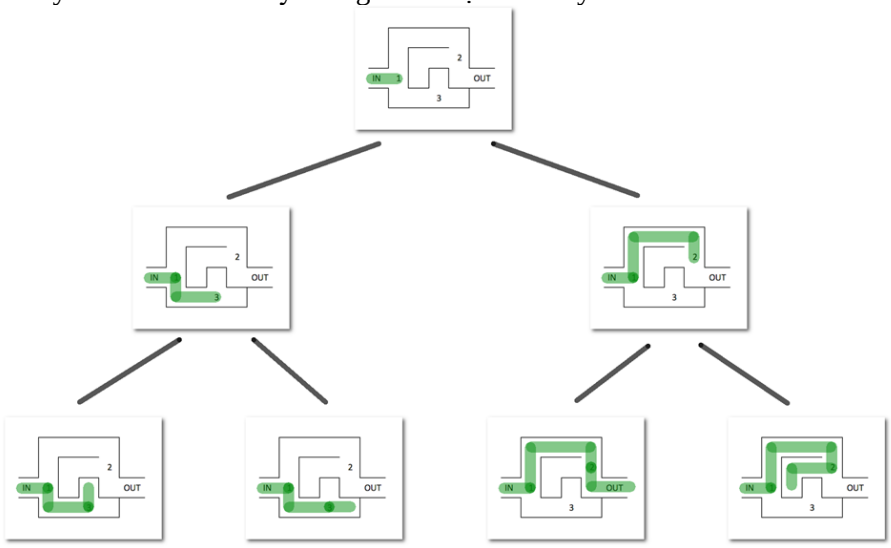
```
1. Nhap sudoku tu ban phim
2. Nhap nhanh sudoku
3. In ket qua sudoku
9. Thoat
Bam phim chuc nang:     2
da nhap xong:
5 3 0  | 0 7 0  | 0 0 0
6 0 0  | 1 9 5  | 0 0 0
0 9 8  | 0 0 0  | 0 6 0
-------------------------
8 0 0  | 0 6 0  | 0 0 3
4 0 0  | 8 0 3  | 0 0 1
7 0 0  | 0 2 0  | 0 0 6
-------------------------
0 6 0  | 0 0 0  | 2 8 0
0 0 0  | 4 1 9  | 0 0 5
0 0 0  | 0 8 0  | 0 7 9
Press any key to continue . . .
```

```
1. Nhap sudoku tu ban phim
2. Nhap nhanh sudoku
3. In ket qua sudoku
9. Thoat
Bam phim chuc nang:     3
ket qua:
5 3 4  | 6 7 8  | 9 1 2
6 7 2  | 1 9 5  | 3 4 8
1 9 8  | 3 4 2  | 5 6 7
-------------------------
8 5 9  | 7 6 1  | 4 2 3
4 2 6  | 8 5 3  | 7 9 1
7 1 3  | 9 2 4  | 8 5 6
-------------------------
9 6 1  | 5 3 7  | 2 8 4
2 8 7  | 4 1 9  | 6 3 5
3 4 5  | 2 8 6  | 1 7 9
Press any key to continue . . .
```

2. Bài toán Mã đi tuần.

2.1. Giới thiệu bài toán mã đi tuần.

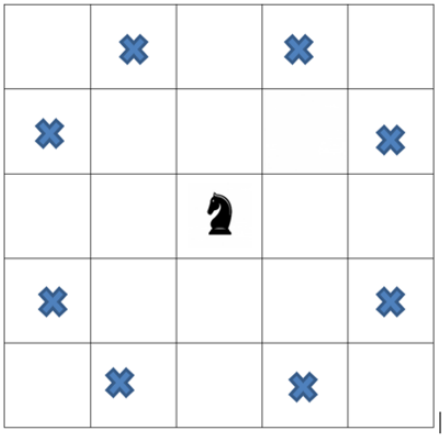
- Mã đi tuần: là bài toán di chuyển một quân cờ một quân mã trên bàn cờ vua (8 x 8).Quân mã được đặt ở một ô trên một bàn cờ trống nó phải di chuyển theo quy tắc của cờ vua để đi qua mỗi ô trên bàn cờ đúng một lần.
-Với bài toán này chúng ta có thể áp dụng thuật toán đệ quy, thuật toán quay lui. Ý tưởng khá đơn giản, khi bạn đứng trước 2 con đường, bạn sẽ chọn 1 đường để đi, nếu đường đó là đường cụt, bạn quay lại vị trí ban đầu và đi đường thứ 2, thuật toán sẽ thử hết những khả năng có thể xảy ra cho đến khi mà nó đạt được mục đích của mình, quá trình này sẽ diễn ra liên tục cho đến khi tất cả các điểm trong bàn cờ được con Mã đi đến.
-Hình này có thể miêu tả rõ ý tưởng của thuật toán này:



- Với ý tưởng như vậy chúng ta sẽ giải quyết vấn đề này như sau:
• Sử dụng thuật toán đệ quy, thuật toán quay lui để đi đến tất cả các vị trí có thể đi trên bàn cờ.

- Nếu con Mã đi vào một vị trí mà nó không thể đi tiếp (tức là những điểm nó có thể đi từ điểm hiện tại đã được đặt chân đến rồi) thì quay lại vị trí trước đó và lựa chọn con đường khác.
- Việc lặp lại này sẽ diễn ra liên tục cho đến khi tất cả các điểm trên bàn cờ được đi tới.

2.2. Giải thuật thiết kế đường đi cho con Mã:



- Nếu bạn biết chơi cờ vua thì hoàn toàn biết rằng con Mã có thể đi chéo 2 nước. Vì thế trong hàm đường đi ta sẽ thiết kế như sau:

- Giả sử x là vị trí theo cột của con Mã, y là vị trí theo dòng của con Mã. Thì 1 trong 2 vị trí này sẽ thay phiên nhau thay đổi 1 và 2 (Bao gồm cả cộng và trừ), một bước đi hợp lệ của quân mã sẽ như sau: |x| + |y| = 3 (Với x,y > 0).
- Khi đó ở một vị trí bất kì quân mã có có 8 đường có thể di chuyển. Chưa xét đến bước đi đó có hợp lệ hay không.
- Các bước đi đó là: (-2, -1), (-2, 1), (-1, -2), (-1, 2), (1, -2), (1, 2), (2, -1), (2, 1)
- Dựa vào tính chất này ta có thể liệt kê nhiều nhất 8 điểm mà một con Mã có thể đi được đến từ một điểm trong bàn cờ.

```
int xMove[8] = { -2,-2,-1,-1, 1, 1, 2, 2 }; //TỌA ĐỘ X ĐC PHÂN TÍCH CHO MÃ
int yMove[8] = { -1, 1,-2, 2,-2, 2,-1, 1 }; //TỌA ĐỘ Y ĐC PHÂN TÍCH CHO MÃ
```

- Ta lưu những vị trí x và y khả dĩ tương ứng vào 2 mảng, Với 1 vị trí tương ứng trong 2 mảng thì đó chính là một khả năng thay đổi vị trí của con Mã.
- Nếu ta thực hiện vòng lặp đồng thời qua 2 mảng này thì con Mã sẽ đi được đến 8 điểm ta vừa nhắc tới.

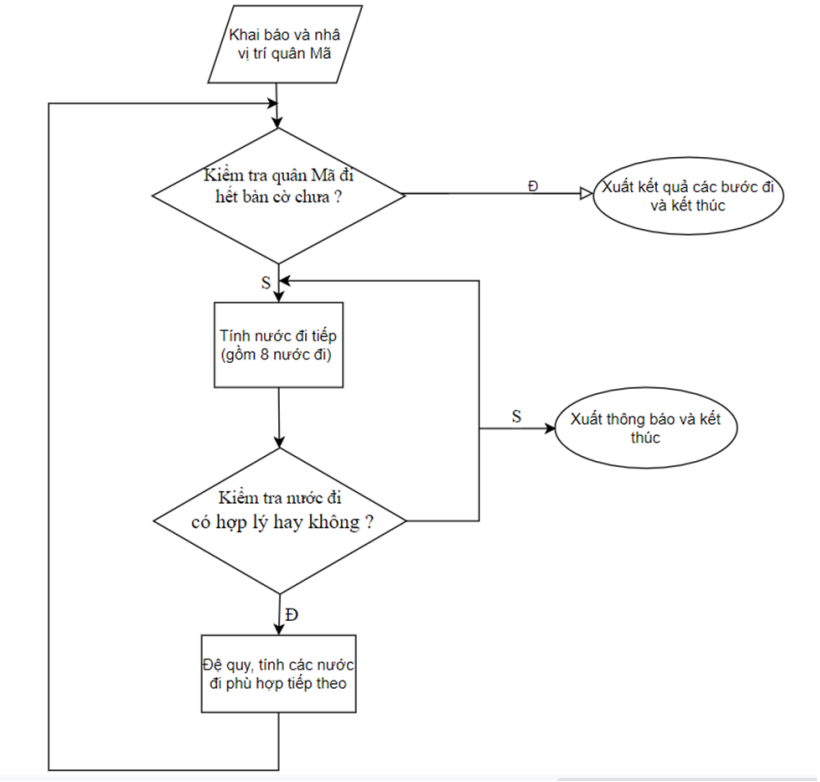
```
for (int i = 0; i < 8; i++) {

    //NẾU CHƯA HẾT BÀN CỜ THÌ ĐI BƯỚC MỚI
    int u = x + xMove[i];   //MÃ SẼ ĐI BƯỚC MỚI TẠI VỊ TRÍ x MỚI
    int v = y + yMove[i];   //MÃ SẼ ĐI BƯỚC MỚI TẠI VỊ TRÍ y MỚI
```

Và để đảm bảo điểm tiếp đến là thuộc bàn cờ ta cần kiểm tra rằng nó nằm trong bàn cờ với hàm sau và nếu đúng thì sẽ thực hiện bước đi

```
//HÀM KIỂM TRA QUÂN MÃ NẰM TRONG BÀN CỜ
bool isSafe(int x, int y) {
    return x >= 0 && y >= 0 && x <= N - 1 && y <= N - 1 && Matrix[x][y] == 0;
}
```

2.3. Sơ đồ giải thuật



2.4. Cách lập trình tạo hàm đi cho Mã

- Khởi tạo hàm mảng ban đầu có giá trị bằng 0. Các tọa độ x, y đã được phân tích ở trên. Khởi tạo giá trị biến đếm Count để gán cho từng bước đi của mã. N là số phần tử của bàn cờ muốn tạo

```
int Matrix[MAX][MAX] = { 0 };    //BAN ĐẦU KHỞI TẠO MẢNG 2 CHIỀU CÓ GIÁ TRỊ BẰNG 0
int xMove[8] = { -2,-2,-1,-1, 1, 1, 2, 2 }; //TỌA ĐỘ X ĐC PHÂN TÍCH CHO MÃ
int yMove[8] = { -1, 1,-2, 2,-2, 2,-1, 1 }; //TỌA ĐỘ Y ĐC PHÂN TÍCH CHO MÃ
int COUNT = 0;  //SỐ BƯỚC MÃ DI CHUYỂN
int N;          //SỐ PHẦN TỬ CỦA BÀN CỜ MUỐN TẠO CHO MỘT HÀNG HOẶC MỘT CỘT
```

- Hàm xuất ra các bước đi quân mã khi có

```
//IN RA CÁC BƯỚC ĐI CỦA MÃ
void PrintArray() {
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            cout << Matrix[i][j] << " ";
        cout << endl;
    }
}
```

- Giải bài toán mã đi tuần bằng đệ quy, thuật toán quay lui để tìm bước đi phù hợp cho quân mã, từ vị trí ban đầu ta sẽ tìm từng bước đi phù hợp nhất và đi đến bước đi đó, còn nếu bước đi không phù hợp sẽ bỏ qua và quay lại tìm bước đi phù hợp. Nếu như qua 8 phép thử từ vòng for trong hàm Move mà củng chưa tìm được đường đi phù hợp thì sẽ trả lại vị trí ban đầu và xuất thông báo không tìm thấy.

```
void Move(int x, int y) {
    ++COUNT;        //TĂNG GIÁ TRỊ BƯỚC ĐI
    Matrix[x][y] = COUNT;//ĐÁNH DẤU ĐÃ ĐI QUA BƯỚC NÀY

        //VÒNG FOR ĐỂ TÌM TẤT CẢ CÁC BƯỚC ĐI PHÙ HỢP VÀ CÁC BƯỚC ĐI THỬ CHO NÓ
    for (int i = 0; i < 8; i++) {
        //KIỂM TRA XEM ĐÃ ĐI HẾT BÀN CỜ CHƯA
        if (COUNT == N * N) {
            cout << "Print steps: \n";
            PrintArray();
            exit(0);    //NẾU IN XONG THÌ KẾT THÚC CHƯƠNG TRÌNH
        }
        //NẾU CHƯA HẾT BÀN CỜ THÌ ĐI BƯỚC MỚI
        int u = x + xMove[i];   //MÃ SẼ ĐI BƯỚC MỚI TẠI VỊ TRÍ x MỚI
        int v = y + yMove[i];   //MÃ SẼ ĐI BƯỚC MỚI TẠI VỊ TRÍ y MỚI
        //KIỂM TRA TÍNH HỢP LỆ THÌ MÃ SẼ DI CHUYỂN
        if (isSafe(u,v) == true)
            Move(u, v);
    }
    //NẾU KHÔNG TÌM THẤY BƯỚC ĐI NÀO PHÙ HỢP THÌ TRỞ LẠI VỊ TRÍ BAN ĐẦU
    --COUNT;
    Matrix[x][y] = 0;
}
```

- Hàm nhập số phần tử bàn cờ muốn tạo và nhập vị trí ban đầu cho quân mã. Nếu như Hàm Move tìm được đường đi hợp lý thì sẽ xuất ra bước đi đó. Còn không tìm được sẽ xuất ra thông báo không tìm được đường phù hợp.

```
void Enter()
{
    cout << "Enter the number of steps : ";
    cin >> N;
    int a, b;
    cout << "Enter initial position.\nx: ";
    cin >> a;
    cout << "y: ";
    cin >> b;
    Move(a, b);
    //NẾU KHÔNG TÌM THẤY ĐƯỜNG ĐI THÌ THÔNG BÁO NHƯ SAU:
    cout << "Don't find !!";
}
```

- Kết quả chương trình:

```
Enter the number of steps : 8
Enter initial position.
x: 0
y: 0
Print steps:
1 12 9 6 3 14 17 20
10 7 2 13 18 21 4 15
31 28 11 8 5 16 19 22
64 25 32 29 36 23 48 45
33 30 27 24 49 46 37 58
26 63 52 35 40 57 44 47
53 34 61 50 55 42 59 38
62 51 54 41 60 39 56 43

C:\Users\MrCHINH\OneDrive - VNU-HCMUS\Desktop\Destop11\TH_OOP\THCTDL&GT\MaDiTuan\Debug\MaDiTuan.exe
ed with code 0.
```

3. Bài toán Xếp Hậu

3.1. Giới thiệu bài toán Xếp Hậu.

- Bài toán N quân hậu là bài toán đặt N quân hậu trên bài cờ vua có kích thước NxN và N quân hậu được đặt sao cho N quân hậu không thể chạm mặt nhau theo hàng ngang, hàng dọc, đường chéo với số ô từ vị trí đứng của quân hậu đến hết bàn cờ.
=> Kết quả là một bàn cờ kích thước N×N và N quân hậu không thể di chuyển

3.2. Giải thuật bài toán Xếp Hậu.

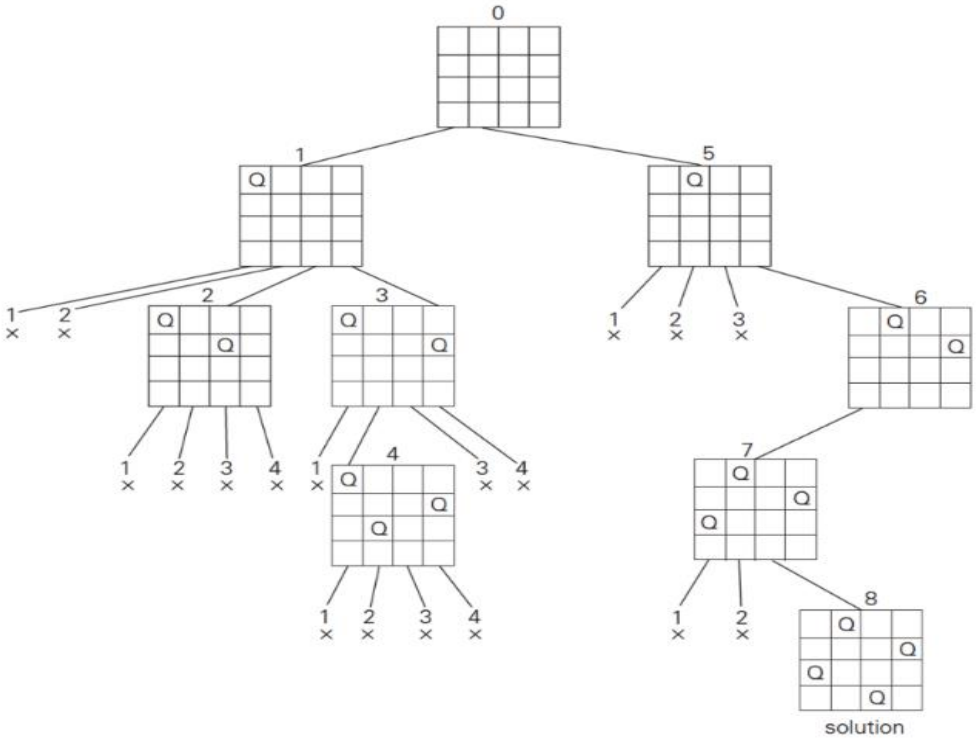
*** Sử dụng thuật toán Backtracking**.

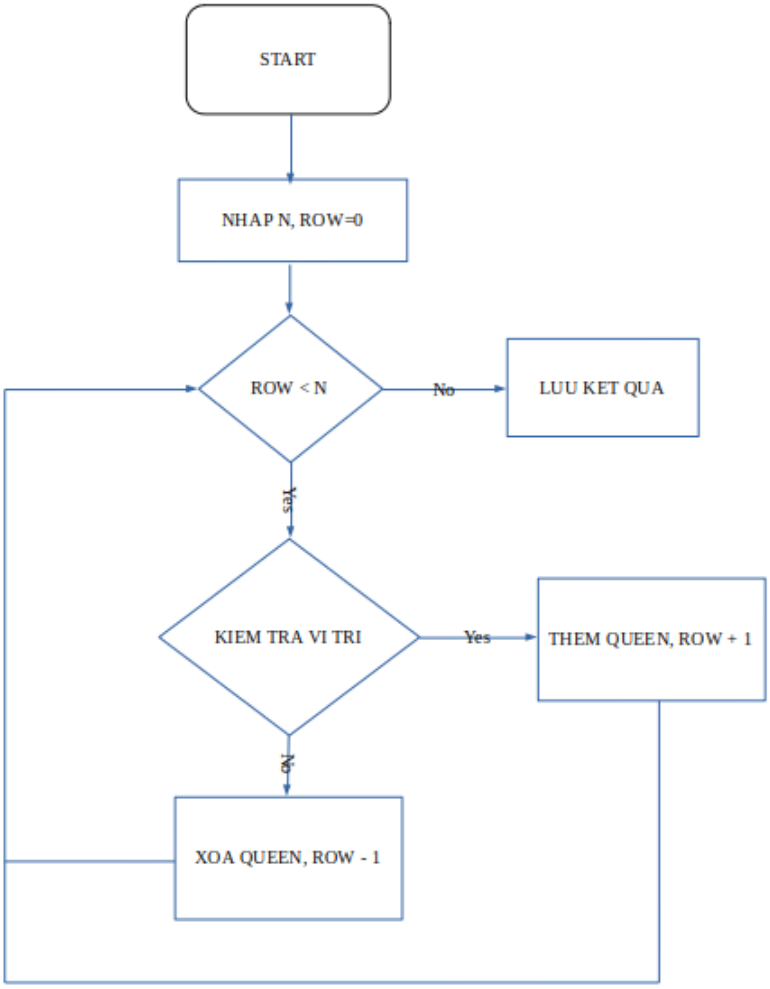
- Xét tất cả các trường hợp đặt quân hậu của thứ nhất (có N trường hợp),

với mỗi trường hợp đặt quân hậu thứ nhất, ta xét các cách đặt quân hậu thứ 2, quận hậu thứ 2 cũng cũng có thể đặt ở N ví trị trên hàng thứ 2, nhưng nó phải né tránh sau cho không bị quân hậu thứ nhất ăn được nó

- Với quân hậu thứ i nó cũng sẽ có N cách đặt, và nó cũng phải né tránh những ô mà i-1 quân hậu trước đó có thể ăn được nó.
- Nếu quân hậu thứ i trong 1 hàng không thể có vị trí phù hợp sẽ quay lại hàng i-1 tăng vị trí cột lên 1.
- Như vậy sẽ sử dụng đệ quy quay lui thay vì sử dụng N vòng lập for lồng nhau.

3.3. Sơ đồ giải thuật bài toán Xếp Hậu





3.4. Cách lập trình bài toán Xếp Hậu

- Sử dụng vector string để lưu trạng thái của bàn cờ và khởi tạo giá trị gồm n phần tử và mỗi phần tử chứa string gồm n ký tự '.', và sẽ thay đổi giá thành 'Q' tại mỗi hàng nếu có vị trí phù hợp
  Và một vector để lưu kết quả các cách thực hiện

```
vector<string> board(n,string(n,'.'));
```

```
vector<vector<string>> ret;
```

a. Kiểm tra vị trí quân hậu:

```
bool is_valid(vector<string> board, int row, int col){
    // kiem tra cot
    for(int i=row;i>=0;i--){
        if(board[i][col]=='Q') return false;
    }
    //kiem tra hang cheo trai
    for(int i=row,j=col;i>=0 && j>=0 ;i--,j--){
        if(board[i][j]=='Q') return false;
    }

    //kiem tra hang cheo phai
    for(int i=row,j=col;i>=0 && j<board.size();i--,j++){
        if(board[i][j]=='Q') return false;
    }
    return true;
}
```

Kiểm tra cột, tiếp theo là kiểm tra hàng chéo trên trái, và hàng chéo trên phải. Nếu phù hợp sẽ đặt 'Q' vào vị trí.

b. Hàm đệ quy quay lui:

```
void dfs(vector<string> &board, int row){

    if(row==board.size()){
        ret.push_back(board);
        return ;
    }
    for(int i=0;i<board.size();i++){
        if(is_valid(board,row,i)){
            board[row][i]='Q';
            dfs(board,row+1);
            board[row][i]='.';
        }
    }


}
```

- Kiểm tra vị trí và đặt 1 quân hậu vào từng hàng nếu hàng nào thỏa mãn thì đến hàng kế tiếp không thì quay lui trở lại hàng trước. kết quả lưu vào vector "ret" khi vị trí hàng cuối cùng thỏa yêu cầu. Và tiếp tục tìm kết quả khác cho đến khi kết thúc đệ quy.
- Kết quả chương trình:

```
nhap N: 4
Cach sap xep 1
.Q..
...Q
Q...
..Q.
Cach sap xep 2
..Q.
Q...
...Q
.Q..
```

```
nhap N: 3
Khong the sap xep
```

4. Bảng Phân Chia Công Việc

STT	Họ và Tên	MSSV	Nhiệm Vụ Công Việc
1	**Võ Văn Chính**	**18200070**	**Bài toán Mã đi tuần** Giải thích chi tiết giải thuật + Code
2	**Nguyễn Ngọc Anh Hào**	**18200095**	**Bài toán Xếp Hậu** Giải thích chi tiết giải thuật + Code
3	**Trần Công Huy**	**18200129**	**Bài toán Sudoku** Giải thích chi tiết giải thuật + Code

Kết Thúc